Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

# Комп’ютерний практикум №3

## Технології паралельних обчислень

**Тема:** Розробка паралельних програм з використанням механізмів синхронізації: синхронізовані методи, локери, спеціальні типи

Виконав
студент групи ІП-11:
Панченко С. В.

Перевірила:
Стеценко І.В.

Київ 2024

# ЗМІСТ

# 1 ЗАВДАННЯ

1. Реалізуйте програмний код, даний у лістингу, та протестуйте його при різних значеннях параметрів. Модифікуйте програму, використовуючи методи управління потоками, так, щоб її робота була завжди коректною. Запропонуйте три різних варіанти управління. 30 балів.
2. Реалізуйте приклад Producer-Consumer application (див. https://docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html ). Модифікуйте масив даних цієї програми, які читаються, у масив чисел заданого розміру (100, 1000 або 5000) та протестуйте програму. Зробіть висновок про правильність роботи програми. 20 балів.
3. Реалізуйте роботу електронного журналу групи, в якому зберігаються оцінки з однієї дисципліни трьох груп студентів. Кожного тижня лектор і його 3 асистенти виставляють оцінки з дисципліни за 100-бальною шкалою. 40 балів.
4. Зробіть висновки про використання методів управління потоками в java. 10 балів.

# 2 ВИКОНАННЯ

## 2.1 Перше завдання

Для виконання першого завдання розробив три засоби синхронізації потоків: з допомогою синхронізованих методів, локерів та атомарних змінних. Розглянемо структуру проєкту:

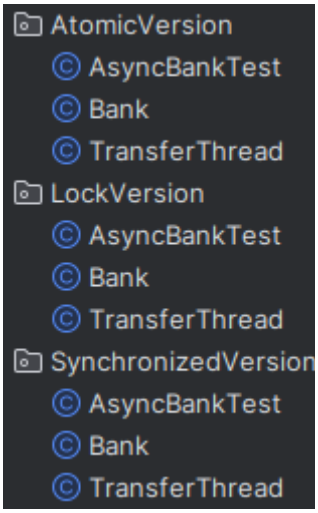


Рисунок 2.1.1 - Структура проєкту

Кожне вирішення задачі має по три аналогічних класи, де відрізняються певні деталі. Покажемо результат виконання коду та зробимо пояснення до нього.

Як версія з локерами, так і синхронізованими методами видає один результат. Бачимо, що сума завжди залишається 100000, оскільки лише один потік в одиницю часу модифікує масив.

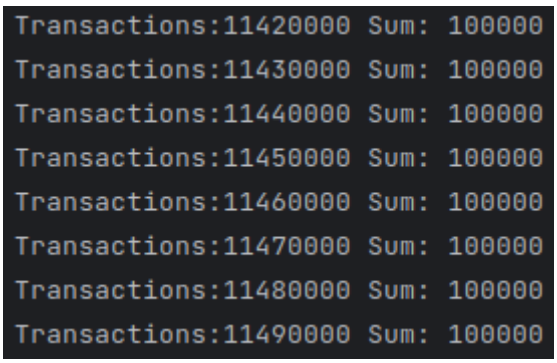


Рисунок 2.1.2 - Результат виконання з локерами або синхронізованими методами

Однак, якщо подивитися на результат виконання з атомічними змінними, то отримаємо, що сума завжди буде повертатися до 100000.

```
Transactions: 12940181 Sum: 100001
Transactions: 12950054 Sum: 99983
Transactions: 12960126 Sum: 100006
Transactions: 12970094 Sum: 100021
Transactions: 12980070 Sum: 100006
Transactions: 12990068 Sum: 100006
Transactions: 13000101 Sum: 99997
Transactions: 13010072 Sum: 100005
```

Рисунок 2.1.3 - Результат виконання з використанням атомарних операцій

Розглянемо код. Атомарні операції гарантують, що лише один потік буде з ними. Однак, між атомарними операціями існує race condition, проте він не впливає ні на що, оскільки кожний потік відніме певне число, а потім додасть таке саме. Таким чином, код виконується швидше, оскільки потоки менше між собою синхронізуються.

```
public void transfer(int from, int to, int amount) {
    accounts.addAndGet( i: from,  delta: -amount);
    accounts.addAndGet( i: to,  delta: amount);
    if(totalTransacts.incrementAndGet() % NUMBER_TEST == 0) test();
}
```

Рисунок 2.1.4 - Код з використанням атомарних операцій

## 2.2 Друге завдання

У другому заданні масив рядків був замінений на масив цілих чисел. Покажемо результат та структуру проєкту.

```
MESSAGE RECEIVED: 1
MESSAGE RECEIVED: 2
MESSAGE RECEIVED: 3
MESSAGE RECEIVED: 4
MESSAGE RECEIVED: 5
MESSAGE RECEIVED: 6
MESSAGE RECEIVED: 7
MESSAGE RECEIVED: 8
MESSAGE RECEIVED: 9
MESSAGE RECEIVED: 10
```

Рисунок 2.2.1 - Результат виконання

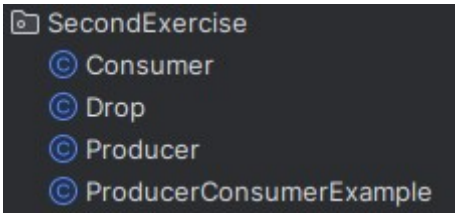


Рисунок 2.2.2 - Структура проєкту

## 2.3 Третє завдання

У третьому завданні синхронізація досягається синхронізованим методом addGrade.

```
public synchronized void addGrade(int grade) {
    grades.add(grade);
}
```

Рисунок 2.3.1 - Синхронізований метод addGrade

Для синхронізованого виведення оцінок групи використав статичний метод printGrades, тобто монітор береться не на об’єкт класу, а на сам клас, тобто може викликатися один статичний синхронізований метод в одиницю часу.

```
public synchronized void printGrades() {
    System.out.println(group.getName() + " " + name + ": ");
    grades.stream().limit( maxSize: grades.size() - 1).toList().forEach( action: g -> System.out.print(g + ","));
    System.out.print(grades.get(grades.size() - 1) + "\n");
}
```

Рисунок 2.3.2 - Синхронізований статичний метод printGrades

```
Group 2 Student 4:
44,5,12,92,25,51,41,82
Group 0 Student 0:
20,21,73,55,3,92,67,23,8
Group 0 Student 1:
72,80,41,8,54,79,6,53,38
Group 0 Student 2:
27,2,58,30,1,25,49,38,41
Group 0 Student 3:
64,50,74,86,92,8,81,43,43
Group 0 Student 4:
82,48,53,47,46,99,49,70,91
Group 1 Student 0:
24,70,38,67,32,69,20,56,38
Group 1 Student 1:
26,24,70,45,26,12,83,47,85
```

Рисунок 2.3.3 - Результат виконання коду

# 3 ВИСНОВОК

Під час лабораторної роботи опрацювали завдання з розробки паралельних програм з використанням механізмів синхронізації. У висновку я можу сказати, що управління потоками у Java зручне, особливо радує наявність synchronized методів, де у C++, на якому програмую найбільше, цього не має.

Однак мушу сказати, що архітектура самої мови Java та наявність garbage collection має свою ціну. Покажемо це на прикладі класів, що реалізують метод Locker. Оскільки C++ має RAII — тобто концепцію конструкторів та деструкторів, то використання локерів( або м'ютексів ) виглядає так:

```
// C++ VERSION
void someFunc() {
	std::lock_guard<std::mutex> lockGuard;
	//do some stuff
}
// JAVA VERSION
class LockerExample {
	private final ReentrantLock lock = new ReentrantLock();
	public void someFunc() {
		// do some stuff
	}
}
```

Оскільки lockGuard — це змінна, то вона при виході з області видимості викличе деструктор, у якому є автоматичне відкриття локера. Тобто програмісту на C++ не треба самому закривати чи відкривати локери, бо за нього це робить конструктор і деструктор. Так само із відкриттям і закриттям файлів тощо. Як не дивно, але garbage collection тут заважає(бо в Java відсутні деструктори), тому пріоритет віддам реалізації на C++.

Щодо атомарних змінних, то поведінка як у C++ так і в Java є аналогічною, наскільки це дозволяє сказати мій досвід. Хоча я не люблю застосовувати atomic, бо мені на багато легше загорнути змінну в синхронізований контекст чи поставити локери.

# ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

*Тексти програмного коду*

(Найменування програми (документа))

*Жорсткий диск*

(Вид носія даних)

(Обсяг програми (документа), арк.)

*Студента групи ІП-113 курсу*

*Панченка С. В*

```
//
./Lab3/Lab3/src/main/java/org/example/FirstExercise/LockVersion/Bank.java

package org.example.FirstExercise.LockVersion;

import java.util.Arrays;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class Bank {
   public static final int NUMBER_TEST = 10000;
   private final int[] accounts;
   private long totalTransacts = 0;
   private final ReentrantLock bankLock = new ReentrantLock();

   public Bank(int n, int initialBalance){
      accounts = new int[n];
      Arrays.fill(accounts, initialBalance);
   }

   public synchronized void transfer(int from, int to, int amount) {
      bankLock.lock();
      accounts[from] -= amount;
      accounts[to] += amount;
      totalTransacts++;
      if(totalTransacts % NUMBER_TEST == 0) {
         test();
      }
      bankLock.unlock();
   }

   public void test() {
```

```
            var sum = Arrays.stream(accounts).sum();
            System.out.println("Transactions:" + totalTransacts + " Sum: " + sum);
        }

        public int size() {
            return accounts.length;
        }
    }


    //
./Lab3/Lab3/src/main/java/org/example/FirstExercise/LockVersion/AsyncBankTest.j
ava

    package org.example.FirstExercise.LockVersion;

    public class AsyncBankTest {

        public static void main(String[] args) {
            var totalAccounts = 10;
            var initialBalance = 10000;
            var b = new Bank(totalAccounts, initialBalance);
            for(var i = 0; i < totalAccounts; i++){
                var t = new TransferThread(b, i, initialBalance);
                t.setPriority(Thread.NORM_PRIORITY + i % 2);
                t.start();
            }
        }
    }


    //
```

```
package org.example.FirstExercise.LockVersion;

class TransferThread extends Thread {
    private final Bank bank;
    private final int fromAccount;
    private final int maxAmount;
    private static final int REPS = 1000;

    public TransferThread(Bank bank, int fromAccount, int maxAmount){
        this.bank = bank;
        this.fromAccount = fromAccount;
        this.maxAmount = maxAmount;
    }

    @Override
    public void run() {
        while(true) {
            for(var i = 0; i < REPS; i++) {
                var toAccount = (int) (bank.size() * Math.random());
                var amount = (int) (maxAmount * Math.random()/REPS);
                bank.transfer(fromAccount, toAccount, amount);
            }
        }
    }
}
```

// ./Lab3/Lab3/src/main/java/org/example/FirstExercise/AtomicVersion/Bank.java

```
package org.example.FirstExercise.AtomicVersion;

import java.util.Arrays;
import java.util.concurrent.atomic.AtomicIntegerArray;
import java.util.concurrent.atomic.AtomicLong;
import java.util.concurrent.atomic.AtomicReference;
import java.util.concurrent.atomic.AtomicReferenceFieldUpdater;

class Bank {
    private static final int NUMBER_TEST = 10000;
      private static final String TRANSACTIONS_SUM = "Transactions: %d
Sum: %d\n";
    private final AtomicIntegerArray accounts;
    private final AtomicLong totalTransacts = new AtomicLong(0);

    public Bank(int n, int initialBalance){
        accounts = new AtomicIntegerArray(n);
        for(var i = 0; i < n; i++) accounts.set(i, initialBalance);
    }

    public static void main(String[] args) {
        var b = new AtomicReference<Bank>(new Bank(10, 10000));
    }

    public void transfer(int from, int to, int amount) {
        accounts.addAndGet(from, -amount);
        accounts.addAndGet(to, amount);
        if(totalTransacts.incrementAndGet() % NUMBER_TEST == 0) test();
    }

    public void test() {
```

```
            var strArray = accounts.toString();
                var sum = Arrays.stream(strArray.substring(1, strArray.length() -
1).split(", "))
                    .map(Integer::parseInt).mapToInt(Integer::intValue).sum();
            System.out.printf(TRANSACTIONS_SUM, totalTransacts.get(), sum);
        }

        public int size() {
            return accounts.length();
        }
    }


    //          ./Lab3/Lab3/src/main/java/org/example/FirstExercise/AtomicVersion/
AsyncBankTest.java

    package org.example.FirstExercise.AtomicVersion;

    public class AsyncBankTest {

        public static void main(String[] args) {
            var totalAccounts = 10;
            var initialBalance = 10000;
            var b = new Bank(totalAccounts, initialBalance);
            for(var i = 0; i < totalAccounts; i++){
                var t = new TransferThread(b, i, initialBalance);
                t.setPriority(Thread.NORM_PRIORITY + i % 2);
                t.start();
            }
        }
    }
```

```
//          ./Lab3/Lab3/src/main/java/org/example/FirstExercise/AtomicVersion/
TransferThread.java

package org.example.FirstExercise.AtomicVersion;

class TransferThread extends Thread {
    private final Bank bank;
    private final int fromAccount;
    private final int maxAmount;
    private static final int REPS = 1000;

    public TransferThread(Bank bank, int fromAccount, int maxAmount){
        this.bank = bank;
        this.fromAccount = fromAccount;
        this.maxAmount = maxAmount;
    }

    @Override
    public void run() {
        while(true) {
            for(var i = 0; i < REPS; i++) {
                var toAccount = (int) (bank.size() * Math.random());
                var amount = (int) (maxAmount * Math.random()/REPS);
                bank.transfer(fromAccount, toAccount, amount);
            }
        }
    }
}


//
```

./Lab3/Lab3/src/main/java/org/example/FirstExercise/SynchronizedVersion/Bank.java

```
package org.example.FirstExercise.SynchronizedVersion;

import java.util.Arrays;

class Bank {
   public static final int NUMBER_TEST = 10000;
   private final int[] accounts;
   private long totalTransacts = 0;

   public Bank(int n, int initialBalance){
      accounts = new int[n];
      Arrays.fill(accounts, initialBalance);
   }

   public synchronized void transfer(int from, int to, int amount) {
      accounts[from] -= amount;
      accounts[to] += amount;
      totalTransacts++;
      if(totalTransacts % NUMBER_TEST == 0) {
         test();
      }
   }

   public void test() {
      var sum = Arrays.stream(accounts).sum();
      System.out.println("Transactions:" + totalTransacts + " Sum: " + sum);
   }

   public int size() {
```

```
            return accounts.length;
        }
    }
```

//
./Lab3/Lab3/src/main/java/org/example/FirstExercise/SynchronizedVersion/
AsyncBankTest.java

```
    package org.example.FirstExercise.SynchronizedVersion;

    public class AsyncBankTest {

        public static void main(String[] args) {
            var totalAccounts = 10;
            var initialBalance = 10000;
            var b = new Bank(totalAccounts, initialBalance);
            for(var i = 0; i < totalAccounts; i++){
                var t = new TransferThread(b, i, initialBalance);
                t.setPriority(Thread.NORM_PRIORITY + i % 2);
                t.start();
            }
        }
    }
```

//
./Lab3/Lab3/src/main/java/org/example/FirstExercise/SynchronizedVersion/
TransferThread.java

```
    package org.example.FirstExercise.SynchronizedVersion;
```

```
class TransferThread extends Thread {
    private final Bank bank;
    private final int fromAccount;
    private final int maxAmount;
    private static final int REPS = 1000;

    public TransferThread(Bank bank, int fromAccount, int maxAmount){
        this.bank = bank;
        this.fromAccount = fromAccount;
        this.maxAmount = maxAmount;
    }

    @Override
    public void run() {
        while(true) {
            for(var i = 0; i < REPS; i++) {
                var toAccount = (int) (bank.size() * Math.random());
                var amount = (int) (maxAmount * Math.random()/REPS);
                bank.transfer(fromAccount, toAccount, amount);
            }
        }
    }
}


// ./Lab3/Lab3/src/main/java/org/example/SecondExercise/Producer.java

package org.example.SecondExercise;

import java.util.Random;

public class Producer implements Runnable {
```

```
    private final Drop drop;

    public Producer(Drop drop) {
        this.drop = drop;
    }

    public void run() {
        int[] importantInfo = new int[1000];
        for (int i = 0; i < importantInfo.length; i++) importantInfo[i] = i + 1;

        Random random = new Random();

        for(var s : importantInfo) {
            drop.put(s);
            try {
                Thread.sleep(random.nextInt(5000));
            } catch(InterruptedException ignored) {
            }
        }
        drop.put(0);
    }
}

//
```

./Lab3/Lab3/src/main/java/org/example/SecondExercise/ProducerConsumerExample.java

```
package org.example.SecondExercise;

public class ProducerConsumerExample {
    public static void main(String[] args) {
        Drop drop = new Drop();
```

```
        (new Thread(new Producer(drop))).start();
        (new Thread(new Consumer(drop))).start();
    }
}
```

```
// ./Lab3/Lab3/src/main/java/org/example/SecondExercise/Drop.java

package org.example.SecondExercise;

public class Drop {
    private int message;
    private boolean empty = true;

    public synchronized int take() {
        while(empty) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        empty = true;
        notifyAll();
        return message;
    }

    public synchronized void put(int message) {
        while(!empty) {
            try {
                wait();
            } catch (InterruptedException e) {}
        }
        empty = false;
```

```
        this.message = message;
        notifyAll();
    }
}
```

```
// ./Lab3/Lab3/src/main/java/org/example/SecondExercise/Consumer.java

package org.example.SecondExercise;

import java.util.Random;

public class Consumer implements Runnable {
    private final Drop drop;

    public Consumer(Drop drop) {
        this.drop = drop;
    }

    public void run() {
        Random random = new Random();
        for(var message = drop.take(); message != 0; message = drop.take()) {
            System.out.format("MESSAGE RECEIVED: %d%n", message);
            try {
                Thread.sleep(random.nextInt(5000));
            } catch (InterruptedException ignored) {}
        }
    }
}
```

```
//
./Lab3/Lab3/src/main/java/org/example/ThirdExercise/StudentsTeacherExample.java

package org.example.ThirdExercise;

public class StudentsTeacherExample {
    private static final int TOTAL_TEACHERS = 4;
    private static final int TOTAL_GROUPS = 3;
    private static final int TOTAL_STUDENTS_IN_GROUP = 5;

    public static void main(String[] args) {
        var groups = new Group[TOTAL_GROUPS];
        for(var i = 0; i < TOTAL_GROUPS; i++) {
                                    groups[i] = new Group("Group " + i,
TOTAL_STUDENTS_IN_GROUP);
        }

        var teachers = new TeacherThread[TOTAL_TEACHERS];
        for(var i = 0; i < TOTAL_TEACHERS; i++) {
            teachers[i] = new TeacherThread(groups);
            teachers[i].start();
        }

        for(var i = 0; i < TOTAL_TEACHERS; i++) {
            try {
                teachers[i].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
```

```
    }
}


// ./Lab3/Lab3/src/main/java/org/example/ThirdExercise/Student.java

package org.example.ThirdExercise;

import java.util.ArrayList;
import java.util.Arrays;

public class Student {
    private final ArrayList<Integer> grades = new ArrayList<>();
    private final String name;
    private final Group group;

    public Student(String name, Group group) {
        this.group = group;
        this.name = name;
    }

    public synchronized void addGrade(int grade) {
        grades.add(grade);
    }

    public synchronized void printGrades() {
        System.out.println(group.getName() + " " + name + ": ");
                grades.stream().limit(grades.size() - 1).toList().forEach(g ->
System.out.print(g + ","));
        System.out.print(grades.get(grades.size() - 1) + "\n");
    }
}
```

```
// ./Lab3/Lab3/src/main/java/org/example/ThirdExercise/Group.java

package org.example.ThirdExercise;

import java.util.ArrayList;
import java.util.Iterator;

public class Group implements Iterable<Student> {
    private final ArrayList<Student> students;
    private final String name;

    public Group(String name, int totalStudents) {
        this.name = name;
        this.students = new ArrayList<>(totalStudents);
        for(var i = 0; i < totalStudents; i++) {
            students.add(new Student("Student " + i, this));
        }
    }

    public String getName() {
        return name;
    }

    public static synchronized void printGrades(Group g) {
        for(var s : g) {
            s.printGrades();
        }
    }

    @Override
```

```
    public Iterator<Student> iterator() {
        return students.iterator();
    }
}


// ./Lab3/Lab3/src/main/java/org/example/ThirdExercise/TeacherThread.java

package org.example.ThirdExercise;

import java.util.Arrays;

public class TeacherThread extends Thread {
    private final Group[] groups;
    public TeacherThread(Group[] groups) {
        this.groups = groups;
    }

    @Override
    public void run() {
        while(true) {
            for(var g : groups) {
                for(var s : g) {
                    s.addGrade((int) (100 * Math.random()));
                }
                Group.printGrades(g);
            }
            try {
                Thread.sleep(4000);
            } catch(InterruptedException e) {
                throw new RuntimeException(e);
            }
```

```
        }
    }
}
```